```
//-----------------------------------------------------------------------------------------------------------
//
// jump_timepos
//
// Description:   Perform a absolute or relative jump in a file
//
// Parameters:    fh          filehandle to access the current files Filedescriptor
//                mode        DVR_JUMP_FROM_CURRENT_POS, DVR_JUMP_FROM_START, DVR_JUMP_FROM_END
//                offset      the amount of 0.01 seconds to jump
//                curtime     current time position in the file, used in relative jumps.
//
// Returns:       DVR_Code
//-----------------------------------------------------------------------------------------------------------
static DVR_Code jump_timepos(int fh, int mode, long offtime, long curtime)
{

  long            seconds;
  long            sectors, deltasectors;
  float           rectime;
  long            content;
  long      sectorsPerSecond;
  ulong     i = 0;
  DVR_Code  ret = DVR_OK;
  NMFS_Code seek_ret = NMFS_OK;

  if(mode == DVR_JUMP_FROM_CURRENT_POS)
      offtime = curtime + offtime;
  else if(mode == DVR_JUMP_FROM_END)
      offtime = FileDesc(fh).rectime + offtime;
  seconds = offtime / 100;                              // make seconds out of the time position
                                                        // (rounded downwards)

  // set the type of content in the file that we want to check on.
  content = (DVR_CONTENT_VIDEO | DVR_CONTENT_AUDIO);
  #define deltatime (long)(((float)seconds * 100 - (float)PlayHeaderBuf[HEADER_REC_TIME]  + (float)
FileDesc(fh).starttime)/ 100)

  rectime = FileDesc(fh).rectime/100;
  if(rectime == 0)
      sectorsPerSecond = (HasVideo(fh) ? 900 : 25);   // no playing time available; just pick some
```

```
                                                    // sane values (see function timepos_set_ratio)
else
    sectorsPerSecond = (long)((float)FileDesc(fh).file_length / rectime);
sectors =  (long)(seconds * sectorsPerSecond);

if(seconds < 3) {
    seek_f_NMFS(fh, NMFS_SEEK_FROM_START, 0, NULL);
}
else if(seconds > rectime - 3) {
    seek_f_NMFS(fh, NMFS_SEEK_FROM_END, -3 * sectorsPerSecond, NULL);
}
else {
    seek_f_NMFS(fh, NMFS_SEEK_FROM_START, sectorsPerSecond * seconds, NULL);
}

ret = dvrGetHeader(fh, PlayHeaderBuf, NMFS_SEEK_RELATIVE, content);    // get the closest header
                                                                       // at this position

while (ret == DVR_OK && (abs((int)deltatime) > 1) && (seek_ret == NMFS_OK)) {
    // iterate until deltatime is less than one second
    tm_wkafter(1);
    deltasectors = deltatime * sectorsPerSecond;          //calculate a next jump.
    if(deltasectors + FileDesc(fh).cur_read >  FileDesc(fh).file_length) {  // too close to end,
                                                                            //  or outside file
        deltasectors =  FileDesc(fh).file_length - FileDesc(fh).cur_read;
    }
    if(deltasectors + FileDesc(fh).cur_read < sectorsPerSecond) {      // too close to beginning
                                                                       //  or outside file
        deltasectors =  sectorsPerSecond - FileDesc(fh).cur_read;
        seek_ret = seek_f_NMFS(fh, NMFS_SEEK_FROM_START, sectorsPerSecond, NULL);
    }
    else {
        seek_ret = seek_f_NMFS(fh, NMFS_SEEK_RELATIVE, deltasectors, NULL); // OK. seek to new position
    }

    if(i++ > 10)                                  // check that we don't get stuck in the loop,
                                                  //  breaks after 10 iterations.
        ret = DVR_ERROR;
    else
    ret = dvrGetHeader(fh, PlayHeaderBuf,NMFS_SEEK_RELATIVE,content );      // get the closest header at
                                                                            //   this position
```

```
  }

  return(ret);
}

/* Explanations

The function 'seek_F_NMFS()' repositions the files read pointer according to the arguments.
```

APPENDIX

Included below is the function 'dvrGetHeader()'. It is used to get the nearest header in the file at the current file position. In this header, which occurs approximately 20 times per second in the file, there is a timestamp. The timestamps are produced at recording time, and have a resolution of 1/100 seconds.

```
//-----------------------------------------------------------------
//
// dvrGetHeader
//
// Description:  Read last data header from given file handle
//
// Parameters:      fh    filehandle
//               pBuf  pointer to an Header buffer array
//               type  type of header to seek for.
//
// Returns:   NMFS_Code
//-----------------------------------------------------------------
static NMFS_Code dvrGetHeader(FileHandle fh, ulong *pBuf, ulong position, ulong type)
{
   NMFS_Code error_code;
   ulong     current_pos;

   if((error_code = seek_f_NMFS(fh, (uchar)position, 0, &current_pos)) != NMFS_OK)
        return(error_code);

   do {
        if((error_code = read_f_NMFS(fh, 1, (uchar*) pBuf)) != 1)
                   return(error_code < 0 ? error_code : NMFS_END_OF_FILE);
        if(position == NMFS_SEEK_FROM_END) {
          seek_f_NMFS(fh, NMFS_SEEK_RELATIVE, -2 , NULL);
            }
   } while( !((*pBuf == DISKID_HEADER) && (*(pBuf+1) & type)) );

   error_code = seek_f_NMFS(fh, NMFS_SEEK_FROM_START, current_pos, NULL);          // reset old position

   return error_code;
}
*/
```